4.1. Embedding ApacheDS into an application

{scrollbar}

Embedding Apache Directory Server into an application

ApacheDS 1.5.5
This site was updated for ApacheDS 1.5.5.
ApacheDS 1.5.7
For an example on how to embed the Apache Directory Server version 1.5.7 please look at the
code present at Example For Embedding version 1.5.7

The idea is to use the server directly without having to communicate with a remote server, eliminating all the network layer. This is one of the reason ADS has been developed in Java: to allow such usage.

We will go through a very simple example step by step to show you how this can be done.

Setting up the environment

We first need to define a base for our project. We use Maven (2.0.9 or above) to build our project, so the structure we will use looks like this:

. |-- pom.xml `-- src `-- main `-- java `-- org `-- apache `-- directory `-- EmbeddedADS.java

You could download both files here:
EmbeddedADS.java
pom.xml

If you don't want to use Maven you need to add the following dependencies to your classpath in order to compile and run this sample.
* apacheds-all-1.5.5.jar
* slf4j-api-1.5.6.jar
* slf4j-log4j12-1.5.6.jar
* log4j-1.2.14.jar

Creating the application

So let's start with the code. It's quite simple: we define a class, add a main() method, and initialize the server, before using it. In the code snippet below, we have removed all the initialization part to keep only the main() method.

package org.apache.directory; /** * A simple example exposing how to embed Apache Directory Server * into an application. * * @author Apache Directory Project * @version Rev, $Date$ */ public class EmbeddedADS { /** The directory service */ private DirectoryService service; ... /** * Creates a new instance of EmbeddedADS. It initializes the directory service. * * @throws Exception If something went wrong */ public EmbeddedADS() throws Exception { init(); } /** * Main class. We just do a lookup on the server to check that it's available. * * @param args Not used. */ public static void main(String[] args) //throws Exception { try { // Create the server EmbeddedADS ads = new EmbeddedADS(); // Read an entry Entry result = ads.service.getAdminSession().lookup(new LdapDN("dc=apache,dc=org")); // And print it if available System.out.println("Found entry : " + result); } catch (Exception e) { // Ok, we have something wrong going on ... e.printStackTrace(); } } }

As you can see, we first initialize the server, and immediately do a lookup to check that we can read an entry from it.

Let's focus on the initialization part now.

Initializing the server

This is done in the **init()** method. It will create the DirectoryService global object (**service**), and create a partition in which we will store the entries.

A partition is a storage point, associated with a DN, root point for this partition. It's a bit like a mounting point on Unix.
We also need a context entry associated to this DN.

Here, we will create the **apache** partition, associated with the 'dc=apache,dc=org' DN.

Here is the code for this method :

... /** * Initialize the server. It creates the partition, adds the index, and * injects the context entries for the created partitions. * * @throws Exception if there were some problems while initializing the system */ private void init() throws Exception { // Initialize the LDAP service service = new DefaultDirectoryService(); // Disable the ChangeLog system service.getChangeLog().setEnabled(false); // Create a new partition named 'apache'. Partition apachePartition = addPartition("apache", "dc=apache,dc=org"); // Index some attributes on the apache partition addIndex(apachePartition, "objectClass", "ou", "uid"); // And start the service service.startup(); // Inject the apache root entry if it does not already exist if (!service.getAdminSession().exists(apachePartition.getSuffixDn())) { LdapDN dnApache = new LdapDN("dc=Apache,dc=Org"); ServerEntry entryApache = service.newEntry(dnApache); entryApache.add("objectClass", "top", "domain", "extensibleObject"); entryApache.add("dc", "Apache"); service.getAdminSession().add(entryApache); } // We are all done ! } ...

We disabled the **ChangeLog** service because it's useless in our case. As you can see, the steps to initialize the server are:

- create a new DirectoryService instance
- add a partition

- add some index
- start the service
- add the context entry

One important point: as the data are remanent, we have to check that the added context entry does not exist already before adding it.

Some helper methods have been used : **addPartition** and **addIndex**. Here they are :

... /** * Add a new partition to the server * * @param partitionId The partition Id * @param partitionDn The partition DN * @return The newly added partition * @throws Exception If the partition can't be added */ private Partition addPartition(String partitionId, String partitionDn) throws Exception { // Create a new partition named 'foo'. Partition partition = new JdbmPartition(); partition.setId(partitionId); partition.setSuffix(partitionDn); service.addPartition(partition); return partition; } /** * Add a new set of index on the given attributes * * @param partition The partition on which we want to add index * @param attrs The list of attributes to index */ private void addIndex(Partition partition, String... attrs) { // Index some attributes on the apache partition HashSet<Index<?, ServerEntry>> indexedAttributes = new HashSet<Index<?, ServerEntry>>(); for (String attribute:attrs) { indexedAttributes.add(new JdbmIndex<String,ServerEntry>(attribute)); } ((JdbmPartition)partition).setIndexedAttributes(indexedAttributes); } ...

That's it! (the attached code will contain the needed imports)

Building the sample

Using Maven this is easy:

$ mvn clean compile [INFO] Scanning for projects... [INFO] ---------- [INFO] Building ApacheDS embedded sample [INFO] task-segment: [clean, compile] [INFO] ---------- [INFO] [clean:clean {execution: default-clean}] [INFO] Deleting directory /tmp/EmbeddedADS/target [INFO] [resources:resources {execution: default-resources}] [WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent! [INFO] skip non existing resourceDirectory /tmp/EmbeddedADS/src/main /resources [INFO] [compiler:compile {execution: default-compile}] [INFO] Compiling 1 source file to /tmp/EmbeddedADS/target/classes [INFO] ---------- [INFO] BUILD SUCCESSFUL [INFO] ---------- [INFO] Total time: 2 seconds [INFO] Finished at: Sat Nov 28 20:02:23 CET 2009 [INFO] Final Memory: 17M/76M [INFO] ----------

The resulting jar can be found in the **target** directory.

Running the sample

Using Maven this is easy:

$ mvn exec:java -Dexec.mainClass=org.apache.directory.EmbeddedADS [INFO] Scanning for projects... [INFO] Searching repository for plugin with prefix: 'exec'. [INFO] ---------- [INFO] Building ApacheDS embedded sample [INFO] task-segment: [exec:java] [INFO] ---------- [INFO] Preparing exec:java [INFO] No goals needed for project - skipping [INFO] [exec:java {execution: default-cli}]

When the main method is run, you should obtain something like :

log4j:WARN No appenders could be found for logger (org.apache.directory.server.schema.registries.DefaultNormalizerRegistry). log4j:WARN Please initialize the log4j system properly. Found entry : ServerEntry dn[n]: dc=Apache,dc=Org objectClass: extensibleObject objectClass: domain objectClass: top dc: Apache

That's it!